AFRL-IF-RS-TR-2004-252
Final Technical Report
September 2004

EXTENDED EXPLOITATION TOOLKIT FOR VIDEO (EXTV)

PAR Government Systems Corporation

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-252 has been reviewed and is approved for publication

APPROVED: /s/

TODD B. HOWLETT
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** SEPTEMBER 2004	**3. REPORT TYPE AND DATES COVERED** Final Jun 02 – Aug 04

4. TITLE AND SUBTITLE EXTENDED EXPLOITATION TOOLKIT FOR VIDEO (EXTV)	**5. FUNDING NUMBERS** C - F30602-02-C-0108 PE - 62702F PR - 3480 TA - EX WU - TV
6. AUTHOR(S) John Krol and Dan Manthey	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) PAR Government Systems Corporation 314 South Jay Street Rome New York 13440	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFEC 525 Brooks Road Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2004-252

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Todd B. Howlett/IFEC/(315) 330-4592/ Todd.Howlett@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

A software toolkit for video data exploitation was developed under this project. This Extended eXploitation Toolkit for Video (EXTV) provides the capability to display and manipulate DoD standard motion imagery (video) along with its metadata. It allows for the creation of NITF standard image products to include a video mosaic or captured frame. A unique video annotation capability was developed and demonstrated under this project. It allows for the creation of a non-destructive annotation layer which can be played over the video. An annotation standard was drafted and presented to the National Geospatial-Intelligence Agency's (NGA) Motion Imagery Standard Board (MISB). Upon further implementation and testing it is expected that this standard will be formally submitted to the MISB for recommendation as a DoD standard. These capabilities were realized through incorporation of EXTV into the Geo*View product. Geo*View is an AFRL and PAR Government Systems Corporation developed imagery viewer and metadata editor, which is JITC certified for NITF.

14. SUBJECT TERMS Motion Imagery, Video Processing, Video Exploitation, Video Annotation	**15. NUMBER OF PAGES** 52
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

This technical report summarizes the work accomplished and information gained during the performance of Contract F30602-02-C-0108 entitled *Extended eXploitation Toolkit for Video (EXTV)*. This contract was sponsored by the Air Force Research Laboratory (AFRL) Multi-Sensor Exploitation Branch (AFRL/IFEC).

AFRL/IFEC has developed a software-based toolkit for video data exploitation. This Extended eXploitation Toolkit for Video (EXTV) supports the capability to capture video frames, mosaic the frames into a composite image, and perform basic image exploitation. The objective of this effort was to augment and enhance the video exploitation capabilities of the EXTV to more robustly support the video exploitation needs of the operational user. This objective was realized through incorporation of the EXTV into the Geo*View product. Geo*View is an imagery viewer and metadata editor that supports the reading, writing, and display of a variety of data formats, including motion imagery. Development of the Geo*View application was originally funded by AFRL/IFEC and has been augmented with enhancements funded under Independent Research and Development (IRAD) by PAR Government Systems Corporation (PGSC). Geo*View has been certified by the Joint Interoperability Test Command (JITC) at "CLEVEL 7"compliant for unpacking of National Imagery Transmission Formatted (NITF) files.

2. SOFTWARE ARCHITECTURE

2.1 Java Media Framework (JMF)

The EXTV is implemented within the Java Media Framework (JMF) and also incorporates additional software as described below. The JMF is an extensible set of software tools (a "framework") for handling time-varying data within Java. It defines mechanisms for playback, capture, and transcoding of time-varying data, and also provides decoders, demultiplexers, and encoders for popular data formats such as MPEG, AVI, and WAV. The framework was originally developed by Sun Microsystems and IBM. Sun now provides implementations that run on any platform that provides a Java Virtual Machine. The source code for these implementations is covered by the Sun Community Source License; therefore, they are publicly available.

The framework consists of modular components ("plug-ins") that provide individual capabilities, such as the decoding of a particular video format. Sun's implementation provides several popular plug-ins, but developers may also create additional components. The framework interconnects these plug-ins to process time-varying data. For formats such as MPEG and AVI there is typically a demultiplexer that separates the video, audio, and metadata components, a decoder for each component, and renderers for the decoded data (see Figure 1).

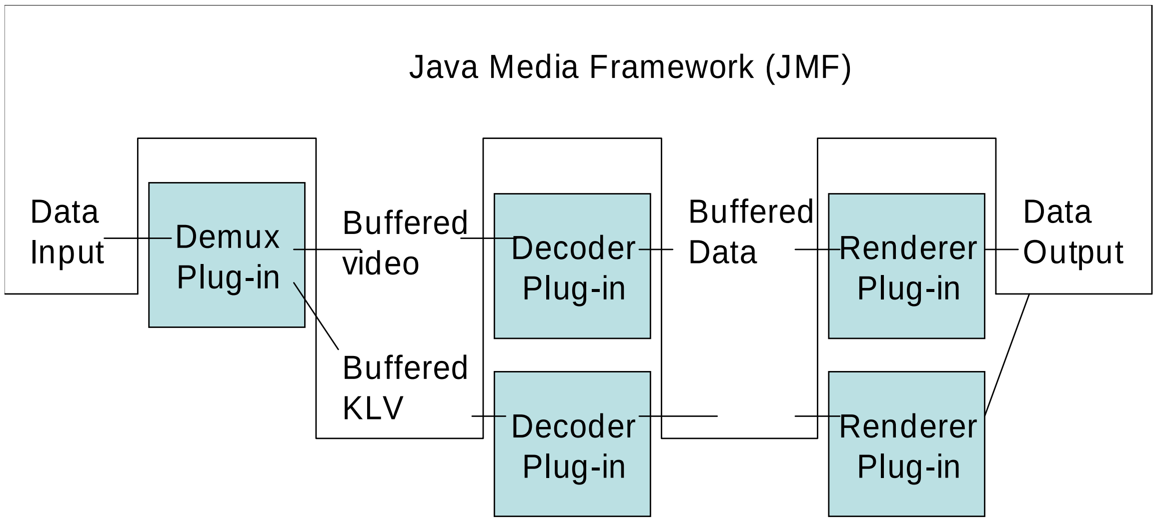


Figure 1 Adding Plug-Ins to JMF

The framework makes heavy use of multiple threads of execution to facilitate the movement of data between the components in the processing chain. It also manages transactions between the components so that usable decoded data is provided to the screen, speakers, or other final destination. The data communicated between each connected pair of components is labeled with its format, allowing the modular components to be independent of one another. The data is communicated in a standardized way, allowing the data to come from a static source (such as a file), a capture device, or even a streaming source.

2.2 New Components

Several new JMF plug-ins were created to allow the EXTV to support Department of Defense (DoD) standards.

MPEG-2 Demultiplexer and Decoder - Separates components from MPEG-2 transport streams if necessary and decodes MPEG-2 video streams. The components provided with the JMF do not handle transport streams and can decode only MPEG-1 video streams.

Key Length Value (KLV) Decoder - Decodes KLV encoded data, such as the geospatial metadata encoded with the output of Unmanned Aerial Vehicles (UAVs).

Several renderers – Used by Geo*View to display decoded data in a way consistent with the overall design of Geo*View.

3. ACCOMPLISHMENTS

The major accomplishments of the EXTV program are presented in the sections that follow. Section 3.1 discusses the implementation of the MPEG-2 decoder and demultiplexer that is now part of EXTV; Section 3.2 discusses the support of the MPEG-2 KLV metadata that has been added to EXTV; Section 3.3 discusses the use of the KLV to add annotation to video; and Section 3.4 discusses additional functionality such as object tracking, NITF support, and mosaicking. Further discussions are provided in Appendix A – MPEG Video; Appendix B – MPEG Systems; Appendix C – KLV; and Appendix D – KLV and Annotations.

3.1 MPEG-2

The Department of Defense (DoD) and Intelligence Communities make extensive use of MPEG-2, so EXTV must be capable of both decoding MPEG-2 video streams and extracting metadata associated with these streams. This capability is provided by the JMF plug-ins for demultiplexing and decoding MPEG-2 streams (defined by ISO 13818) that were created under this effort.

The MPEG-2 standard specifies a method of encoding video and other streams (such as audio) as well as a method for linking those streams together into "system streams". Section 3.1.1 describes the support that has been added to EXTV for decoding video streams, while Section 3.1.2 discusses the support for system streams.

3.1.1 MPEG Video

The video decoding function added to the EXTV has been implemented in the form of a stand-alone library for decoding MPEG video, written in strictly-conforming C. It may be used with video data contained in MPEG system streams or other formats combining multiple streams, such as QuickTime. This library is then attached to the JMF as a plug-in component.

The decoder consists of a state machine that decodes particular portions of a video stream, and then providing the decoded data to user-specified callbacks. There is additional detail on the structure and interfaces to the decoder in Appendix A. Section A.1 provides background on the structure of MPEG video; Section A.2 discusses the interface to the decoder library; Section A.3

discusses the state machine; and Section A.4 defines the callbacks. Section A.5 discusses the supported and unsupported features of MPEG-2 video.

3.1.2 MPEG Systems

A coded MPEG video stream is insufficient to carry all information associated with some data sources. For instance, it cannot carry the audio data associated with the motion video of a movie, nor can it carry the metadata that the DoD sometimes associates with motion imagery surveillance. Therefore, Part 1 of the MPEG-2 standard (ISO 13818-1) provides mechanisms for compositing several encoded “elementary streams” together into a single sequential stream of data called a “system stream”. These mechanisms are, for the most part, independent of the encoding of the streams being composited, allowing not only MPEG-2 video and audio streams, but also any other sort of data, such as DoD metadata.

Since these mechanisms encode multiple streams together with no modification to the coding of the individual streams, they are said to “multiplex” the streams. EXTV’s ability to decode MPEG-2 system streams is provided by a “demultiplexer”. This demultiplexer, like the MPEG video decoder, is implemented by a stand-alone library written in strictly conforming C and attached to the JMF as a plug-in.

Further detailed information on MPEG Systems is provided in Appendix B. Section B.1 discusses the library and system streams; Section B.2 discusses the demultiplexer interface; and Section B.3 discusses the demultiplexer main function.

3.2 MPEG-2 and KLV

The MPEG-2 Transport Stream can contain a stream of Key Length Value (KLV) metadata in addition to the multiplexed video. This KLV data can be used to encapsulate any data that is to be associated with the video stream. Section 3.3 discusses the use of KLV data by EXTV to encapsulate annotations into a data extension segment (DES) of a NITF. Decoding those annotations requires reading the DES of the NITF and then parsing the data.

Appendix C provides details of the support of KLV encoding and decoding as they are implemented in EXTV. Section C.1 discusses the Society of Motion Picture and Television

Engineers (SMPTE) standard for using KLV, and Section C.2 discusses the techniques used in EXTV to parse the KLV.

3.3 KLV and Annotations

Annotations that are added to video are often simple in their structure and repetitive from frame to frame. Storing them as video frames would be inefficient even with the use of video compression algorithms. A more appropriate approach to adding annotation to an MPEG stream is to encode the annotation with KLV using conventions that result in a compact representation of the annotation. In the EXTV, the video has associated with it a set of annotations. Each annotation consists of some data, such as a piece of text or a computer graphics metafile (CGM) file describing some feature of the video. This data is associated with the subset of the video's frames for which the data is valid. For each video frame that occurs in that subset, the annotation has an associated position within the frame. Video annotations may be encoded within a NITF document by placing the video stream in a data extension segment and the stream of encoded messages in another data extension segment.

The techniques used in EXTV to encode annotations using KLV are described in detail in Appendix D. Section D.1 discusses the mechanisms used to represent the annotations. Section D.2 discusses the encoding of the information describing the annotation into KLV. Section D.3 shows how annotations are presented for viewing.

3.4 Additional EXTV Functions

This effort has considered functions in addition to annotation. They include operator aids such as voice annotation and object tracking as well as support of storage of images in NITF, associating image positions with geographic coordinates, and forming mosaics from image sequences.

3.4.1 Voice Annotations and Tracking

Research was performed on the possible ways to use voice recognition software to annotate video. A range of speech recognition products were investigated from those developed at AFRL to off-the-shelf offerings. Due to the need for a limited vocabulary, support of a large range of speech engines providing swap-out capability, and ease of integration, CloudGarden’s implementation of the Java Speech API (JSAPI) was used. CloudGarden supports a wide variety of Windows Operating Systems, 98, ME, NT4, 2000, XP. Other products, such as IBM's Speech for Java, require IBM's Via Voice product and is supported only on Linux, Windows 95, and NT.

3.4.1.1 Support for Large Range of Speech Engines

All fully compliant Speech Application Programming Interface (SAPI 4.0, 4.1, and SAPI 5) recognizers and synthesizers should be accessible with this implementation, making a large number of speech engines compatible with Geo*View. In virtually all situations, the engine type is transparent to the implementation but, when needed, the engine type (i.e., SAPI 4 or 5) can be identified.

3.4.1.2 Full Implementation of Sun's JSAPI 1.0 Specification

Sun's Java Speech Application Programming Interface (JSAPI) specification was implemented, allowing recognition grammars to be specified in Sun's Java Speech Grammar Format. Use of the JSAPI allows for a quick integration cycle with products that are written in Java.

This implementation was used to allow the user to point to a region of interest (ROI) on a video being viewed and to select the ROI with the mouse. The user can then speak a word or phrase to make the associated annotation appear on the video and track the ROI until the ROI leaves the display. The user has the ability to select which annotation (i.e., text, icon, sound, etc.) is associated with each word or phrase. The user can add and remove entries from the vocabulary and associate icons or sounds with an entry using the graphical user interface (GUI).

The use of voice recognition in annotating video worked fairly well within Geo*View. The use of a quality microphone is a necessity to minimize the number of misunderstood words or phrases. Some success was achieved in creating the ability to associate an image with an entry in the vocabulary and then have that image displayed when the word was spoken. During testing, there were instances when the annotation displayed did not match the spoken word. Occasionally the spoken word was not recognized at all and no annotation was displayed. However, the use of voice recognition software for voice-to-text annotation was found to be particularly useful because it permits the user to concentrate on the real-time video display while verbalizing the user's annotation. This will improve the analyst's exploitation productivity and reliability because attention need not be diverted by having to use the keyboard. It is PGSC's recommendation that this technology be further investigated to identify possible solutions to the discovered limitations.

3.4.2 Object Tracking

Objects may be tracked either manually or automatically. In manual tracking, the user adds an annotation to a region of interest (ROI) and then drags the mouse along while the video plays, thus manually moving the ROI. In automatic tracking the user adds the annotation to a ROI , and the position of the ROI is then automatically updated by a tracking algorithm.

The current algorithm performs a very rudimentary tracking. The red-green-blue (RGB) pixel values for the ROI are saved and used in searching for a match in subsequent frames. For each successive frame, the positions of all annotations are updated. Each existing ROI is updated with its new position found in the next frame. This is done by searching near a prediction of the next position based on an extrapolation of a least-squares-fit line determined from previous locations. If it is determined that a match was found, the annotation will be added to the frame; otherwise, the annotation will be removed from the display.

There are some known limitations to this tracking algorithm:

- There is currently no "outline" capability of the actual ROI. A box drawn to surround the ROI is the only annotation available. The extra area in each box that does not belong to the ROI often degrades tracking performance, particularly when the object is moving with regard to the background.

- If an ROI is selected in a very smooth region, the algorithm will not find good matches and the box will tend to wander. While this is not easily corrected, users will not typically have a need to select a big area with little detail since such a region is not of much interest.

- If, as in some clips, there is a great deal of noise on one or more of the edges (such as narrow black or green lines), then the tracker may become confused. Objects that the user thinks should be disappearing off the edge of the screen may have their associated box remain visible. This occurs because the box resists crossing the noisy region that obscures a tracked target.

- If there are one or more frames of pure noise (due to bit errors or a corrupt source), the program will almost certainly lose track.

Even with these known limitations, tracking performed well within Geo*View. It is recommended that different tracking algorithms be pursued to address the shortcomings of this implementation.

3.4.3 NITF Support

Input to the EXTV requires the output of a still image or a mosaicked image of a video. Geo*View can now provide this output by saving an image in the NITF 2.1 file format. Additionally, the video from which a mosaic was constructed may be stored with its mosaic by insertion into a data extension segment (DES) of the NITF file. An appropriate viewer, such as Geo*View, can then display the video along with its mosaic.

3.4.4 Geo-Mapping

Geo*View has been augmented with geo-mapping capabilities that include the ability to display latitude and longitude coordinates corresponding to the position of the mouse cursor over the map. The latitude and longitude is displayed as long as the video being viewed has KLV metadata associated with it. Another feature that has been developed is the ability to tie a video with KLV metadata to a map and plot the target movement on the map, using the KLV target longitude and latitude, as the video is played.

3.4.5 Mosaicking

The baseline EXTV mosaicking capability was integrated into the EXTV component of Geo*View. Geo*View obtains uncompressed video frames from the JMF and uses them to produce a mosaic. The uncompressed images are then overlaid to form a single image containing all images in the video. This resultant image is formed incrementally, growing as each uncompressed image extends the region covered. The resulting mosaic will be displayed by

Geo*View. Once the mosaic has been created, it may be saved in one of several different file formats.

The mosaicking algorithm currently used is quite rudimentary, having been provided by AFRL/IFEC primarily as a place-holder to be updated later with a more sophisticated capability. This algorithm uses only two parameters in its model of motion between video frames, accounting only for translation. This model is insufficient for all but the simplest of videos.

4. DEMONSTRATIONS AND SUPPORT

During the performance of the EXTV effort, numerous demonstrations of Geo*View were provided to AFRL/IFEC and interested organizations visiting AFRL/IFEC. Demonstrations and technical assistance were also provided at the following meetings:

August 2002: Meeting at Langley AFB, representatives from AFC2ISRC/IN, ACC/INYR, and 480IG/DOO were present for a demonstration of annotations and Geo*View.

November 2002: Motion Imagery Standards Board (MISB) and the Interoperability Working Group (IWG) working session at Science Application International Corporation (SAIC). The annotation capability being developed was made into a study item for the IWG.

June 2003: Motion Imagery Standards Board (MISB) and the Metadata Working Group working session at SAIC. An update briefing and demonstration of the annotation capability was provided. This study was transferred from the Interoperability Working Group (IWG) to the Metadata Working Group.

September 2003: Meeting with AFRL, MediaWare Solutions, PGSC, and SAIC at SAIC Dayton to discuss incorporating a video annotation capability into the Distributed Common Ground Station (DCGS) Video Processing Capability (VPC) system.]

November 2003: Motion Imagery Standards Board (MISB) and the Interoperability Working Group (IWG) working session at Science Application International Corporation (SAIC). The IWG presented a demonstration of the Community Motion Imagery Library (CMIL). The CMIL currently uses a web interface, but plans to make the system Geospatial and Imagery Access Services (GIAS) compliant in the future.

April 2004: Motion Imagery Standards Board (MISB) and the Interoperability Working Group (IWG) working session at Science Application International Corporation (SAIC). A presentation was given by representatives from JITC and there was a great deal of discussion on advanced compression and H.264 (High Def) format video.

5. CONCLUSIONS AND RECOMMENDATIONS

Under the Extended eXploitation Toolkit for Video (EXTV) program, a Java-based software framework was developed to integrate a set of capabilities and video exploitation tools for MPEG-2 video. These capabilities include the support for viewing and editing MPEG-2 video files, inserting a newly encoded annotation stream into the DES portion of a NITF file, mosaic creation, geo-mapping, KLV metadata support, and video annotation.

The Geo*View application was successfully used as the demonstration tool for technology with the AFRL/IFEC developed eXploitation Toolkit for Video. The application development focused on augmenting the capabilities of the EXTV to more robustly support the exploitation needs of the operational user. Geo*View, which was developed for AFRL/IFEC, is a Java-based application that provides capabilities to display and edit images and metadata stored in NITF.

Simple mosaicking and tracking algorithms were shown to be usable with a video viewer. Tracked regions of interest and other video annotations can be conveniently displayed, overlain on a video as it plays. These annotations can be encoded or decoded in existing standards, such as NITF and MPEG-2. Video annotation is a new and valuable addition to the tools available for the exploitation of video. The annotation capability developed under this effort is the basis for a new effort sponsored by AFRL to add a prototype real-time annotation capability to the Distributed Common Ground System (DCGS) video processing application.

The use of voice recognition software for voice-to-text annotation was explored and found to be particularly useful because it permits the user to concentrate on the real-time video display while verbalizing his annotation. This will improve the analyst's exploitation productivity and reliability because attention need not be diverted by having to use the keyboard. The use of a standard API, Java Speech Application Program Interface (JSAPI), allows the flexibility to plug different implementations of the API into Geo*View. This will allow for integration of future advancements in speech technology.

As part of Geo*View, a metadata management capability now exists allowing the user to view metadata, modify editable fields, and save changes. These enhancements to Geo*View represent a significant contribution in supporting the video exploitation needs of the operational user.

As a result of the successful development performed under the EXTV program, the National Geospatial-Intelligence Agency (NGA) selected Geo*View as the video viewing application of choice for the Image Exploitation Capability (IEC) Program.

PAR Government Systems Corporation recommends that the following enhancements be considered for augmentation to the EXTV:

- Support for new standards (MPEG-4 Part 10 and HDTV)
- Improved motion imagery annotation (e.g. auto-fill and real-time support)
- Multi-media product generation using Material eXchange Format (MXF) and Advanced Authoring Format (AAF)
- Cueing techniques of potential targets for the analyst
- Motional analysis tools (calculate speed and direction based on metadata and frame rates)
- Real-time reporting from user input and metadata
- Precise positioning
- Tools to assist in bomb damage assessment (BDA) analysis
- Registration (video to video, video to other sensors)
- Exploitation of motion infra-red (IR) data
- Automatic enhancement techniques
- Cueing techniques from live electronic intelligence (ELINT)
- New tracking algorithm development
- Higher order parameter model for mosaicking

APPENDIX A – MPEG VIDEO

A.1 Structure of MPEG Video

Encoded MPEG video is comprised of several layers. The entire video stream is called a “sequence”, within which there is a series of “pictures” composed of several “slices”, each of which is made up of a sequence of “macroblocks”. Each layer consists of a header, which specifies some information about that layer, and one or more elements beneath the layer, with macroblocks containing encoded video sample values.

Each picture contains the information for a single frame of video or one half of a frame. Since adjacent frames of a typical video are largely similar, some of the pictures are encoded using information from previously encoded pictures, allowing better compression. However, if each picture were to require all information from previous pictures, any error encountered in one picture would damage all pictures following it. Therefore some pictures are encoded without using previous information. Specifically, there are three kinds of picture encodings, “I-pictures”, “P-pictures”, and “B-pictures”, which use information from zero, one, and two previous pictures, respectively. The MPEG stream is structured so that after an I-picture and the following I-picture or P-picture have been decoded correctly, there can remain no damage due to errors in pictures prior to the I-picture.

Similarly, the slices into which a picture is divided constitute independently coded regions so that an error in one slice cannot affect any other slice within the same picture. The macroblocks in a slice consist of a sequence of Huffman codes, which provide good compression but are fragile with respect to errors. The insertion, deletion, or change of a single bit is likely to cause the misinterpretation of all following Huffman codes. The separation of macroblocks into slices allows such misinterpretation to be confined to only the slice that contains the error.

A.2 Decoder Interface

The MPEG video decoder library consists of a single C function that takes four arguments: (1) the complete state of a currently active decoder; (2) a set of user-specified callback functions; (3) a buffer containing bits of the video stream to be decoded; and (4) a Boolean value specifying whether the decoder is to perform only a single action or as many as it can without encountering an error. This last argument allows the decoder to be readily used either as a central system component or a small element of a system that can only be allotted a small amount of time. The function returns a code denoting any error that occurred in decoding.

The separation of all decoder states into a separate object allows the decoder library to be safe for concurrent use by multiple decoders, i.e., each decoder need only guarantee that no single state object be simultaneously accessed by multiple threads of execution. The state object contains the values of all fields recovered from the decoded video stream, some useful values derived from these, and the information needed for proper functioning of the decoder. In addition, it stores the video frame currently being decoded and the decoded video frames that are currently available for the decoding of P- and B-pictures. The structure of the decoder state object is especially convenient in that, with the exception of the stored video frames, it is a single contiguous C object requiring no special memory management. The memory used by the stored frames may be provided via a user-specified callback function that is called once when stream decoding first begins.

As a video stream is decoded, the given user-specified functions are called in order either to report the occurrence of various structures within the stream, such as slices, or to provide the decoded results, such as a video frame. Each such callback function, if provided, is passed the following information:

The decoder state object - Provides the user full access to the decoder, allowing great flexibility. The inclusion of event-specific data also allows the uninterested user to avoid direct knowledge of the decoder’s internal structure.

The set of callback functions – Allows any user callback to change the subsequent effect that the decoder has on the user’s program.

The stream buffer - Allows the user to consume or modify the undecoded stream, which may be particularly useful if the MPEG video stream is carried within a larger structure.

Any additional information that is relevant to the reported event.

The user may specify two special callback functions, which do not report coded structures or data. The first of these is passed the stream buffer whenever more bits of the encoded stream must be made available in order for the decoder to proceed. This allows the decoder function to recover from a lack of data rather than returning a code indicating that lack. The second function is called whenever an error is encountered in decoding, thus possibly allowing the user to recover from the error rather than merely returning the code denoting the error. The structure of MPEG video streams often allows errors to be confined to a small bounded region simply by skipping ahead to an easily-found point in the encoded stream.

The data from the MPEG stream to be decoded is provided in a separate object (rather than in a format such as a raw buffer of input bytes) in order to allow the decoder to use only a portion of the input at any given time, and to report which bytes have been used. Not only does this obviate any ability to "unuse" bytes that a user callback function might want to change, but it also allows the decoder to decode related portions of the stream together without concern that an incomplete portion might be provided in one call to the decoder and then be unavailable when the remainder of the portion is provided in the next call. This in turn allows sufficient portions of the stream to be processed together so that the decoder need only maintain a small amount of information about what structures of the stream have been consumed.

Before any structure or portion of a structure within the video stream is consumed, the decoder examines the buffered stream data to determine if the whole portion is available. If not, then no part of that portion is consumed, and more data is requested. This prevents the expensive decoding of structures that are not usable until more of the encoded stream is available, and reduces the frequency of data transfer to and from the stream buffer, thus often reducing the overall time spent copying that data.

A.3 Decoder State Machine

The decoding process is complicated beyond what is suggested by the four layers described in Section A.1.

Pictures may be grouped into "groups of pictures" (GOPs), which start with their own header. These GOPs are for the most part a legacy of MPEG-2's origins in the MPEG-1 standard, and are no longer important.

The header for a sequence may be replicated, appearing between pictures as well as before the first picture of the sequence. The confusion of concatenated sequences is removed by a marker at the end of each sequence.

The use of B-pictures significantly complicates the coding of video, since a picture can only depend on previously coded pictures. However, B-pictures use one picture temporally prior to itself and another that is temporally subsequent to itself. It is therefore necessary to code pictures in a different order than the order in which they are to be displayed.

In order to handle these complications, the decoder uses a finite state machine to manage the various parts of the stream that it must decode.   There are six major states, some of which can be divided into several similar states. Figure 2 shows the state-transition diagram for this state machine. Each arc is labeled with the portion of an MPEG video stream that is consumed in the state transition. In a few cases it is not consumed but is only detected as the next portion present in the stream. Each arc is also labeled with the user-specified callback function that is called to report the change of state.

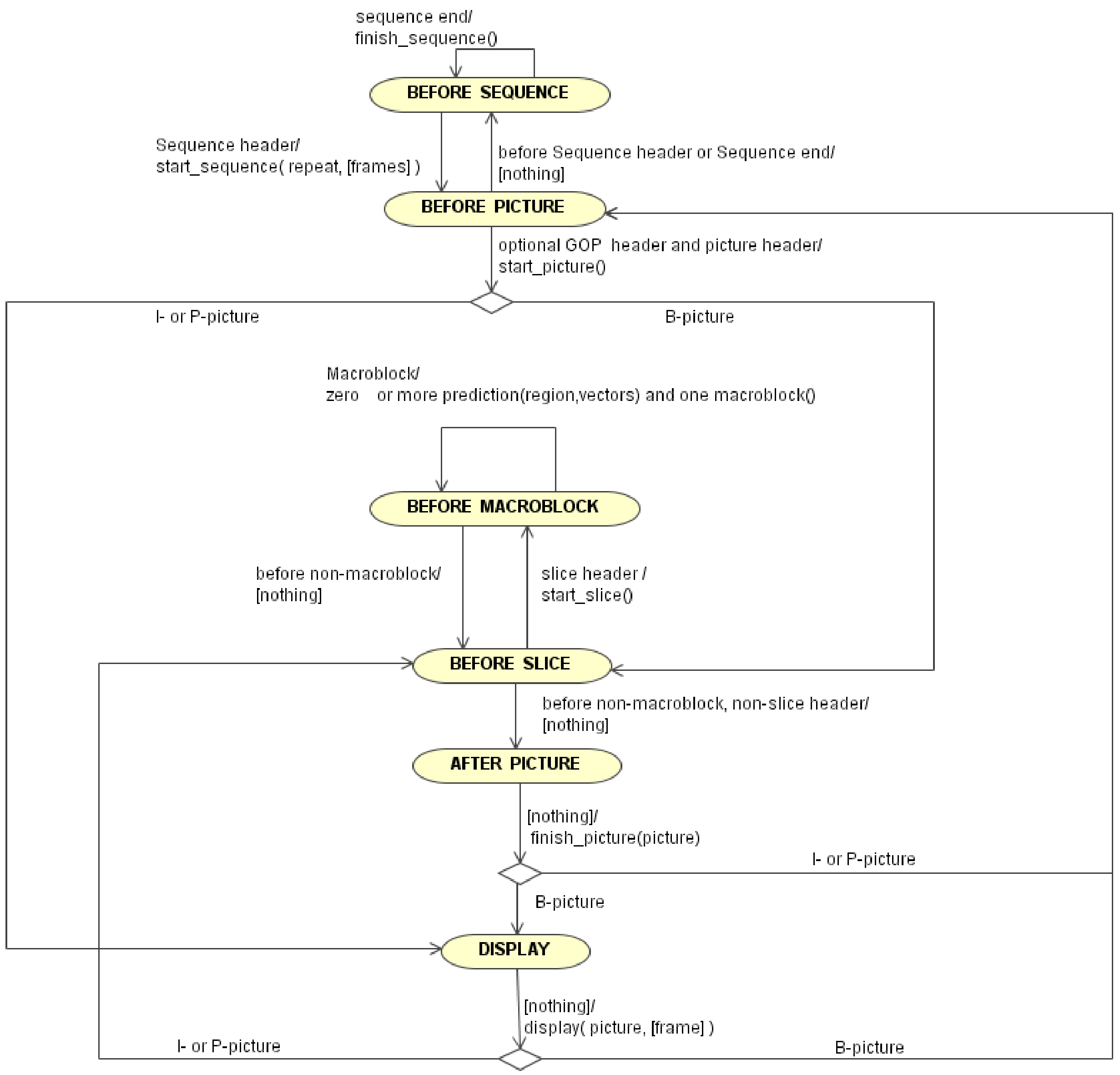


Figure 2 State Transition Diagram

A.4 Decoder Callbacks

As explained above and shown in Figure 2, the user may specify functions that are used to report structures and results. The available functions are listed below. For each of these, the user may specify no function to be called.

- **start_sequence(repeat, [frames])** is called whenever a sequence header is consumed. This can occur either at the beginning of the sequence or when the header is repeated within the stream. In addition to the usual arguments, this callback function is passed a boolean value specifying whether the header is a repeated header or the actual start of a stream. If it is the start of a stream, then this function is also passed a data structure for storing memory that will be used to store video frames.

- **finish_sequence()** is called whenever the end of a stream is reached.

- **start_picture()** is called whenever the header for a picture is consumed.

- **finish_picture(picture)** is called after all of the slices of a particular picture are consumed. In addition to the usual arguments, this function is passed the decoded data of that picture.

- **display(picture, [frame])** is called whenever a decoded picture is the next to occur in temporal order. Note that this function will receive pictures in a different order than the **finish_picture()** function. In addition to the usual arguments, the decoded data for the picture is passed. If the picture completed a video frame (which need not be the case, since a single picture may contain only half of a frame), then the decoded data for the whole frame is passed to this function.

- **start_slice()** is called whenever the header for a slice is consumed. Since this function is rarely specified by the user, the decoder library may be set to never call this function. This is useful since the library always checks for which function the user has specified, and slices occur very frequently. Setting the library to never call this function removes the need to perform that check.

- **macroblock()** is called whenever a macroblock is consumed. As with **start_slice(),** the library may be set to never call this function. In addition to the usual arguments, this function is passed an additional state object that describes the macroblock that has been consumed. This additional state object is not part of the decoder state object because the decoder only consumes whole macroblocks and thus never needs to store any part of one.

- **prediction(region, vectors)** is called whenever previously decoded frame data is used to decode some data of the current frame. As with **start_slice(),** the library may be set to never call this function and, as with **macroblock()**, this function is passed a state object describing the macroblock whose data depends on the previously decoded data. This function is also passed a description of what part of the picture depends on the previously decoded data, and a set of vectors describing how that data was used to decode the current picture.

- **user_action()** is called whenever another user-specified callback function places the decoder in a special state that causes it to call this function. The exact use of this feature is determined by the user.

- **fill_stream(amount)** is called whenever the decoder requires more data to continue decoding, as described in Section A.2, *Decoder Interface*. In addition to the usual arguments, this function is passed the amount of data that is needed to continue. Note that only a minimum amount of data is passed at a time and the decoder may ask for additional data multiple times in order to complete the process.

- **error(error)** is called whenever the decoder encounters an error, as described in Section A.2, *Decoder Interface*, with an additional argument specifying the error that has occurred.

A.5 Supported and Unsupported Features of MPEG Video

The MPEG-2 video standard (ISO 13818-2) is quite complex and provides for several categories of compliance. This categorization consists of two parts; the set of MPEG video structures that are handled by a decoder, called the supported "profile", and the numerical values of various parameters for which the decoder works, called the supported "level." The EXTV decoder library fully supports the “Main Profile” of MPEG-2 and partially supports the “4:2:2” profile, meaning that it can decode any individual MPEG video stream, but does not support the MPEG features for composing multiple streams into a higher-resolution video. The library is designed to use available resources so that it has no predetermined limits to the parameter values that it accepts. However, on any particular computer there is some maximum complexity of video that can be handled at an acceptable rate of decoding.

APPENDIX B – MPEG SYSTEMS

B.1 Structure of System Streams

The details of this library and the system streams it demultiplexes are described below.

B.1.1 PES Packetization

The MPEG-2 standard provides two means of compositing multiple streams, both of which first break each elementary stream into smaller parts. Each part is preceded by a header, forming a single packet of data called a “Packetized Elementary Stream (PES) packet”. The sequence of these packets is composited to form a single data stream. The header of each PES packet can contain a number of fields. One field may be an integer, called a “timestamp”, that denotes the amount of time elapsed from some particular reference time, called the “timebase”. This allows different portions of the elementary stream to be associated with different times.

In the simplest case, the concatenation of all the packets of a single PES is considered a valid system stream, called a “PES stream”. A PES stream cannot be considered to truly composite elementary streams, since only a single elementary stream is used.

B.1.2 Programs

Several elementary streams may be grouped together into a single “program”, in which case the times stored in the headers of the corresponding PES packets all denote the amount of time elapsed from a single reference time (i.e., all of the PES streams share a single timebase).

The set of PES packets making up a single program may be interleaved and concatenated, along with some additional headers, to form a “program stream”. Although the PES packets for each elementary stream must occur in their original order, packets from the various streams may be interleaved freely. Generally, packets marked with similar times are placed near one another, thus allowing two streams with time-dependent data to be synchronized.

B.1.3 Transport

The MPEG-2 standard now allows several programs to be interleaved into a single stream, thus combining many elementary streams. The PES packets that form each elementary stream are again broken up into smaller parts, each of which is preceded by a header to form a “transport stream packet”. Additional information describing the programs is also placed in transport stream packets, and all packets are interleaved and concatenated to form a single “transport stream”.

An error that occurs in one transport packet may affect the use of data in following transport packets. The second layer of packetization is not particularly useful for isolating errors. There are, however, two benefits to this second layer.

First, each transport stream packet’s header begins with a specific pattern of eight bits, which is not typical of most patterns generated by bit errors. This allows a decoder to begin reading a transport stream at an arbitrary location and to determine with reasonable certainty where within that stream the various transport stream packets begin and end. There is no such ability available within program streams.

Second, each transport stream packet is of a small, fixed size. This can make it quite efficient to transmit successfully over many network connections. Unfortunately, the small packet size increases the fraction of the data stream that is composed of header data rather than elementary stream data.

It is possible to place only a single program in a transport stream and to convert the program stream header information to the format used in a transport stream. Similarly, it is possible to extract the data of a single program from a multi-program transport stream and to convert the appropriate header information. Thus, for a single program, either a program stream or a transport stream may be used, and the appropriateness of each is determined by the relative need for compactness of data and robustness to errors in transmission.

B.2 Demultiplexer Interface

The MPEG demultiplexer library consists of two major components and a number of minor accessories. One major component is used to demultiplex PES streams and program streams, while the other is used to demultiplex transport streams. The minor accessories are used to manage certain data structures that are used by the two components. The component used to demultiplex PES streams and program streams is not currently used by Geo*View.

Similar to the video decoder library, many of the functions that make up the demultiplexer library, in addition to whatever arguments are specific to their operation, take a small set of well known arguments, an object that contains the complete state of the demultiplexer, a set of user-specified callback functions, another set of user-specified callback functions used only to report errors, and an object used to access data from the stream being demultiplexed. The exact structure of the state object and the set of callback functions depends on what type of system stream being demultiplexed. All functions use the same structure for the set of error callbacks and the stream access object.

The state object contains the values of any fields recovered from the demultiplexed stream. Which fields are actually recovered depends on the exact use of the library. The state object also stores additional information needed for proper operation of the library. The separation of all demultiplexer states into a separate object allows the demultiplexer library to be safe for concurrent use by multiple demultiplexers, with each demultiplexer using a distinct state object. This separation is somewhat more complex than in the case of the video decoder library, since the demultiplexer state consists of several distinct regions of memory, some of which could be referenced by otherwise distinct state objects. The library cannot be expected to behave safely or predictably if any such sharing occurs. Many of the operations of the library require the addition or removal of regions of memory from the state object. The management of regions of memory is often a complex task which may require significant control by the user. Therefore, part of the user-specified callback function is a distinct object providing functions that the library uses to acquire and relinquish memory. Consequently, some library functions that do not take the full set of callback functions do take the full set of memory management functions.

As a system stream is demultiplexed, user-specified functions are called in order to report various information decoded from the stream or interesting structures within the stream. This decoded information includes some types of errors that can occur in the stream without impairing the operation of the demultiplexer. Similarly, more major errors are reported by calling the given error callback functions. Each callback function, if provided, is passed the demultiplexer state object, the set of callback functions, and the set of error callback functions, as well as any additional information that is relevant to the reported information. The inclusion of the demultiplexer state object provides the user full access to the demultiplexer, and allows great flexibility. The inclusion of event-specific data also allows the uninterested user to avoid direct knowledge of the demultiplexer's internal operation.

The data from the MPEG system stream to be demultiplexed is provided via a separate object in order to allow the decoder to manage which portion of the stream is consumed. The demultiplexer passes the stream access object to one of a few library functions in order to obtain a reference to a buffer of memory which holds an amount of data that is next available in the stream. After some amount of this data is consumed, another function is called to report that data is not to be given again to the demultiplexer. This mechanism obviates the return of the data to the stream access object and allows the demultiplexer to access a sufficient portion of the stream to process together, requiring only a small amount of information to be maintained about which structures of the stream have been consumed.

B.2.1 Handlers

In addition to the callback functions explicitly passed to the library functions, the demultiplexer state object can reference functions and associated data objects (together called "handlers") that are called to provide user access to data recovered from the elementary streams, along with any information from headers associated with that data. The most important data callback function is passed information that is extracted from PES packets, and receives data from elementary streams as well as the timestamps associated with that data. A separate data callback function is passed information that has been broken into parts called "sections". These sections are most common in transport streams, but can also occur in PES streams and program streams. A demultiplexer of transport streams can reference a handler that is passed complete transport packets.

A handler's function is passed the demultiplexer state object, the set of user-specified callback functions (including error functions), the data object associated with the handler, and any information specific to the data that is to be handled. This allows great flexibility in the actions a handler may take upon receiving data.

B.2.2 Management of Other Data

The demultiplexer maintains a number of data structures whose data is decoded from the system stream. These structures are made available to the user through convenient access via the demultiplexer state object. As the stream is decoded, these data structures may change. These changes are reported to the user via the given callback functions. Since only some of these data structures are used to control how the stream is demultiplexed, it is not always necessary for the demultiplexer to decode the associated information. In this case, the information may not be decoded and the user's callback functions will not be called. Conversely, it may be appropriate for the user to change the values of these data structures directly in order to affect the way that the stream is demultiplexed, as well as the way that the library provides functions to perform such changes. Which data structures are actually decoded is controlled by library functions.

Some of the data structures that change over the course of the stream may have their new values coded in the stream prior to those new values taking effect, in which case the demultiplexer may also decode these expected values and report them to the user. When either a new value or the next expected new value is decoded from the stream, a callback function is called. The additional arguments to that function always have the same organization, although their type depends on the information decoded. The callback function is always passed the newly decoded data and whether that new data is a new value or a value expected to be used later. If the decoded value is new, then the old value (if available) is also passed, as is the value expected to be used next, if it differs from the actual new value. If the decoded value is a value expected to be used later and it differs from a previously decoded value expected to be used later, but not yet decoded as the new value, then the previously expected value is also passed to the callback function.

Although what data structures are maintained depends on the kind of system stream being demultiplexed, there are two significant data structures that are used in all cases, which are described in the following two section.

B.2.3 Program Maps

For each program in a system stream there may be a map that specifies what elementary streams are part of that program and may associate some additional data with each of those streams.

PES streams cannot contain such a map. However, since they are very similar to program streams in structure, the demultiplexer treats PES streams as though they were program streams with only a single elementary stream. The structure of transport streams requires that each and every program in the stream has an associated map, although it is possible for transport streams to contain elementary streams that are not associated with any program.

B.2.4 Clocks

For each program in a system stream, as described in Section B.1, *Structure of System Streams*, there must be a single timebase. Since MPEG is intended to allow data to be transmitted with carefully controlled temporal information, it is useful to include in a system stream not only timestamps of individual PES packets, but also a description of the time to which each byte of the system stream corresponds. This is done by including an integer, called a “clock reference”, in certain header information. The clock reference denotes the amount of time elapsed from an associated timebase and may be used to either conceptually or in actuality construct the time, relative to the timebase, of any portion of the system stream. This association of times can be called a “clock”, so that each program not only has an associated timebase, but also a clock corresponding to that timebase.

In transport streams, the program map for a program also indicates which clock is associated with that program. This allows multiple programs to specify the same clock and thus share a common timebase.

B.3 Demultiplexer Function

Regardless of whether the system stream being demultiplexed is a PES stream, a program stream, or a transport stream, all demultiplexing of that stream type is controlled by calling a single function. The function to demultiplex PES streams and the function to demultiplex program streams both take identical demultiplexer state objects and sets of callback functions. They differ only by which header information is examined to determine clock references. However, the function to demultiplex transport streams is significantly different in that it takes a different type of state object and a different set of callback functions.

B.3.1 Program Stream Demultiplexer

The program stream demultiplexer, which is also used to demultiplex PES streams, maintains only a small amount of control information, a single program map, the clock for the one program, an additional data structure called the “program directory” (which can be used to speed up non-sequential access to the system stream), and a set of handlers of PES packets (which receive all of the elementary stream data from the program in the stream). The maintenance of each of these data structures can be activated or deactivated.

B.3.2 Callbacks

As explained Section B.2.2, *Management of Other Data*, given functions are called when certain data is decoded from the system stream as described below:

- **new_id(id)** is called whenever a PES packet from an elementary stream, not previously encountered in the stream, is first decoded. The number identifying the new elementary stream is passed as an additional argument to this function.

- **clock_reference(new clock reference, old clock reference)** is called whenever a clock reference is decoded from the system stream. It is passed as additional arguments the new clock reference and the most recent previous clock reference.

- **timebase(timebase)** is called whenever the timebase of the program in the system stream is determined and is passed the newly determined timebase as a clock reference of the associated clock. This function is generally only called once, near the beginning of the program stream.

- **set_pm(expected, new program map, [old program map], [previously expected new program map])** is called whenever a new program map is completely decoded. The additional arguments are the new map and other information as described in Section B.2.2, *Management of Other Data,* for changing data structures.

- **set_directory(expected, new program directory, [old program directory], [previously expected program directory])** is called whenever a new program directory is completely decoded. The additional arguments are the new directory and other information as described in Section B.2.2, *Management of Other Data,* for changing data structures.

B.3.3 Transport Stream Demultiplexer

The transport stream demultiplexer maintains a small amount of control information, a "program association table" that describes what programs are present in the transport stream, a "transport stream descriptor table" that describes the transport stream, a program map, and an associated clock for each program. The maintenance of each of these data structures can be activated or deactivated.

Each transport stream packet identifies the associated elementary stream or control information by a unique number, called its "PID" (Packet Identifier). A counter is located in the header of each transport packet that is incremented in each successive packet containing the same PID. The demultiplexer records the counter for each PID and will indicate an error when a packet contains an unexpected counter value.

For each PID, the demultiplexer records the method that will be used to process all packets with that specific PID. Most commonly, the data from transport stream packets with a particular PID is used to reconstruct the original PES packet. The reconstructed PES packet is then processed by handlers as described in Section B.3.1, *Program Stream Demultiplexer*. The packets may also be decoded to form sections, whose data is passed to handlers. The user may provide handlers that process the transport stream packets directly.

B.3.4 Callbacks

As explained in Section B.2.2, *Management of Other Data*, given functions are called when certain data is decoded from the system stream.

- **new_pid(pid)** is called whenever a transport stream packet with a PID not previously encountered is first decoded. The new PID is passed as an additional argument to this function.

- **discontinuity(pid, expected, actual)** is called whenever the counter for a particular PID is decoded to have a value other than that expected by the decoder. The PID itself, the expected counter value, and the decoded counter value are passed as additional arguments.

- **clock_reference(new clock reference, old clock reference, displacement)** is called whenever a clock reference is decoded from the transport stream. It is passed as an additional argument to the new clock reference, the most recent previous clock reference, and an amount by which the timebase for the clock has changed. This last value is most often zero, but may be non-zero as indicated by certain fields in the header of the transport stream packet from which the clock reference was decoded.

- **set_pat(expected, new program association table, [old program association table], [previously expected new program association table])** is called whenever a new program association table is decoded. Its additional arguments are the new table and other information as described in Section B.2.2, *Management of Other Data,* for changing data structures.

- **set_tsdt(expected, new transport stream descriptor table, [old transport stream descriptor table], [previously expected new transport stream descriptor table])** is called whenever a new transport stream descriptor table is decoded. Its additional arguments are the new table and other information as described in Section B.2.2, *Management of Other Data,* for changing data structures.

- **set_pmt(expected, new program map, [old program map], [previously expected new program map])** is called whenever a new program map is completely decoded. Its additional arguments are the new map and other information as described in Section B.2.2, *Management of Other Data,* for changing data structures. Information about which program's map has been updated is passed with the usual arguments.

APPENDIX C - KLV

C.1 SMPTE Encoding of KLV

The Society of Motion Picture and Television Engineers (SMPTE) standard 336M-2001(KLV) describes an octet-level data encoding protocol for representing data items and data groups. The standard defines a KLV triplet as a data interchange protocol for data items where the key identifies the data by means of a Universal Label (UL), the length specifies the length of the data, and the value is the data itself. The KLV protocol provides a common interchange for all compliant applications regardless of the method of implementation or transport.

The standard also provides methods for combining associated KLV triplets in data sets where the set of KLV triplets is itself coded with KLV data coding protocol. Such sets can be coded in either full form (universal sets) or in one of four increasingly bit-efficient forms referred to as global sets, local sets, variable-length packs, and fixed-length packs.

Universal sets are used to construct logical groupings of data elements and other KLV encoded items. A universal set uses the full KLV coding construct throughout, marking each contained value with a UL key and a length.

Global sets are defined like universal sets, but share a common key header. Each contained value is marked with a length and an abbreviated form of the UL key. However, the full UL key can be reconstructed without additional information.

Local sets are defined per universal sets, but denote UL keys with a short local tag whose meaning is defined only within the context of the local set. A separate definition is required to define the meaning of the local tags.

Variable-length packs eliminate the use of UL keys and local tags for individual elements within the group. Each element contains a length and associated value. To recover the individual elements of the pack, a separate definition is needed to determine the UL keys of the elements.

Fixed-length packs are defined like variable-length packs, but lengths are not stored with the elements and the separate definition must supply these as well.

C.2 PARSING of KLV

KLV metadata can be parsed into its components as shown in Figure 3. The data is read in until a valid header (0x06 0x0E 0x2B 0x34) is found. The registry category designator is then read to determine the coding of the individual items. The registry designator is used to determine the length of the value of the item and to verify that the item can be read.

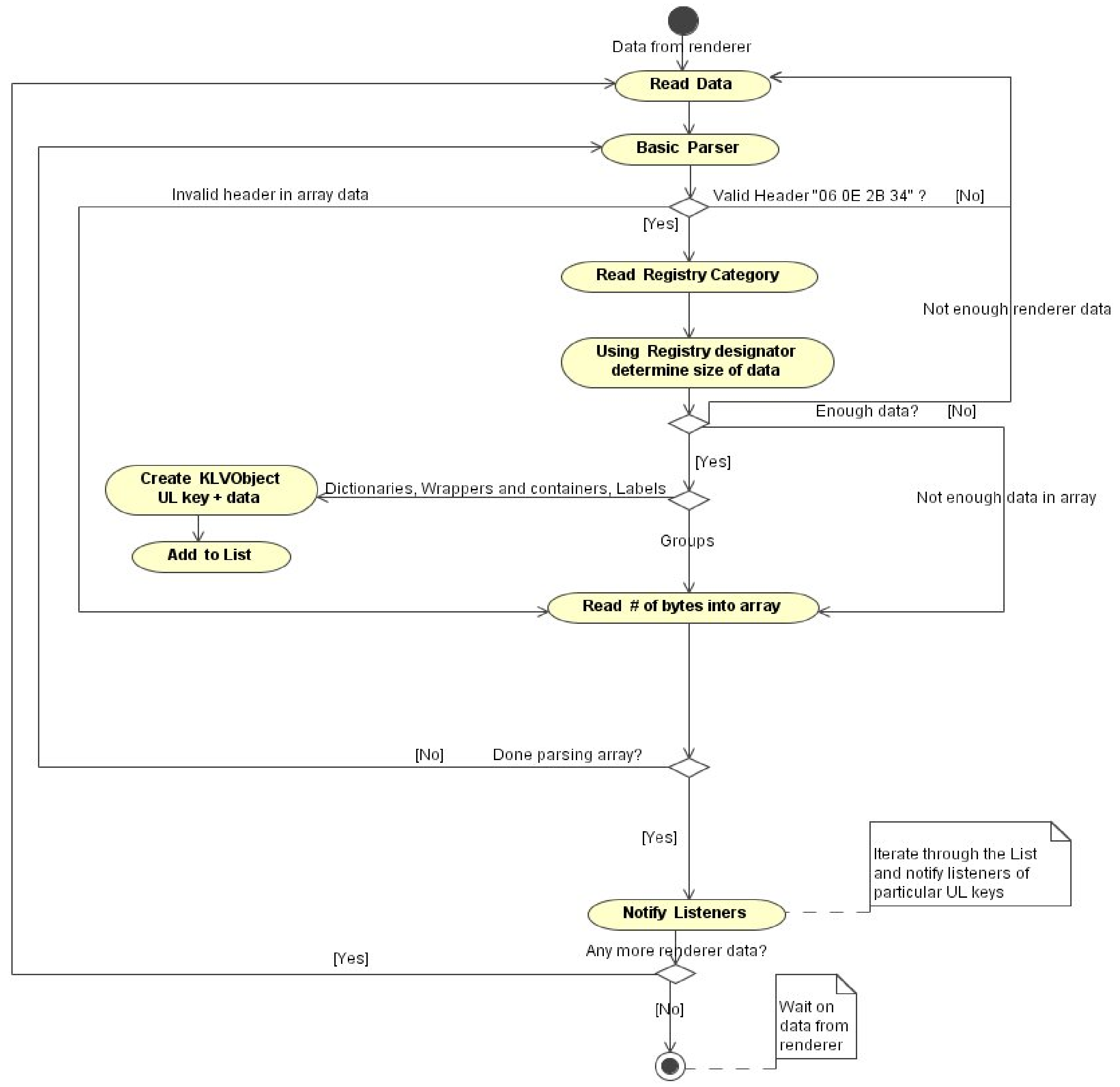

Data from renderer
Read Data
Basic Parser
Invalid header in array data
Valid Header "06 0E 2B 34" ?
[No]
[Yes]
Read Registry Category
Not enough renderer data
Using Registry designator determine size of data
Enough data?
[No]
[Yes]
Create KLVObject UL key + data
Dictionaries, Wrappers and containers, Labels
Not enough data in array
Add to List
Groups
Read # of bytes into array
[No]
Done parsing array?
[Yes]
Iterate through the List and notify listeners of particular UL keys
Notify Listeners
Any more renderer data?
[Yes]
[No]
Wait on data from renderer

Figure 3 KLV Data Parse Component

For global sets, the registry designator indicates how the length field is coded as shown in Table 1 (from the SMPTE Standard 336M-2001).

Table 1 Global Set Length Field Coding

Octet 6 value	Length fields	Description
0x02	BER short or long	Any length
0x22	1 octet	Length up to 2^8-1
0x42	2 octets	Length up to 2^{16}-1
0x62	4 octets	Length up to 2^{32} -1

For local sets, the registry designator indicates how the length field is coded as shown in Table 2 (from the SMPTE Standard 336M-2001).

Table 2 Local Set Length Field Coding

Octet 6 value	Length fields	Local tag fields length	Description
0x03 0x13 0x1B	BER short or long BER short or long BER short or long	1 octet 2 octets 4 octets	Any length
0x23 0x33 0x3B	1 octet 1 octet 1 octet	1 octet 2 octets 4 octets	Length up to 2^8-1
0x43 0x53 0x5B	2 octets 2 octets 2 octets	1 octet 2 octets 4 octets	Length up to 2^{16}-1
0x63 0x73 0x7B	4 octets 4 octets 4 octets	1 octet 2 octets 4 octets	Length up to 2^{32} -1

For variable-length packs, the registry designator indicates how the length field is coded as shown in Table 3 (from the SMPTE Standard 336M-2001).

Table 3 Variable-Length Pack Length Field Coding

Octet 6 value	Length fields	Description
0x04	BER short or long	Default
0x24	1 octet	Length up to 2^8-1
0x44	2 octets	Length up to 2^{16}-1
0x64	4 octets	Length up to 2^{32} -1

In the KLV coding protocol, the value of the length field is encoded using the basic encoding rules (BER) for either the short form or long form encoding of length octets specified in ISO/IEC 8825-1, paragraph 8.1.3. For all other entries not mentioned above, the length fields will be coded as per BER notation with long or short form as required.

In the short form, the length octets shall consist of a single octet in which bit 8 is zero and bits 7 to 1 encode the number of octets in the contents (value) octets (which may be zero), as an unsigned binary integer with bit 7 as the most significant bit. The short form for length encoding is used whenever the data value length is less than 128 octets.

In the long form, the length octets shall consist of an initial octet and one or more subsequent octets. The initial octet is encoded with bit 7 set to one; bits 6 to 0 shall encode the number of subsequent octets in the length octets, as an unsigned binary integer with bit 7 as the most significant bit. Bits 7 to 0 of the first subsequent octet, followed by bits 7 to 0 of the second subsequent octet, followed in turn by bits 7 to 0 of each further octet up to and including the last subsequent octet, shall be the encoding of an unsigned binary integer equal to the number of octets in the value field, with bit 7 of the first subsequent octet as the most significant bit.

If all the data has been read in and the length exceeds the available data, then the data should be discarded. If the data is determined to be a single item entry, the value can then be handled as required by the application. If the item is a set of data, the data will be parsed into individual items based on their registry category and registry designator. In our implementation we create a

KLV object that is a container object for the UL key and the value or UL key and a list of KLV objects for each entry. All parties interested in a particular KLV item designator can register with the parser. They will be notified when requested data has been parsed.

APPENDIX D – KLV AND ANNOTATIONS

D.1 Video Annotation

Video annotation is the ability to dynamically annotate situations and record attributes along with time information of the video being viewed. An operator is primarily interested in documenting changes in a scene across multiple frames. This ability can be greatly enhanced by the use of annotations.

An annotated video is associated with a set of annotations. Each annotation consists of data, such as a piece of text or a CGM file, which describes some feature of the video. This data is associated with the subset of the video's frames for which the data is valid. For each video frame that occurs in that subset, the annotation has an associated position within the frame.

Video annotations were developed based on the user's requirement for associating the annotation information with a particular frame of a video, knowing when to display or remove an annotation, and saving the annotation information for use with a particular video. This annotation information could be stored for each frame of the video, but this might be too data intensive. To minimize the amount of data that is being transferred, a messaging scheme, shown in Figure 4, has been implemented to facilitate the handling of annotations.

Based on these annotation records, the application can determine when to add, move, and remove annotations from the video.

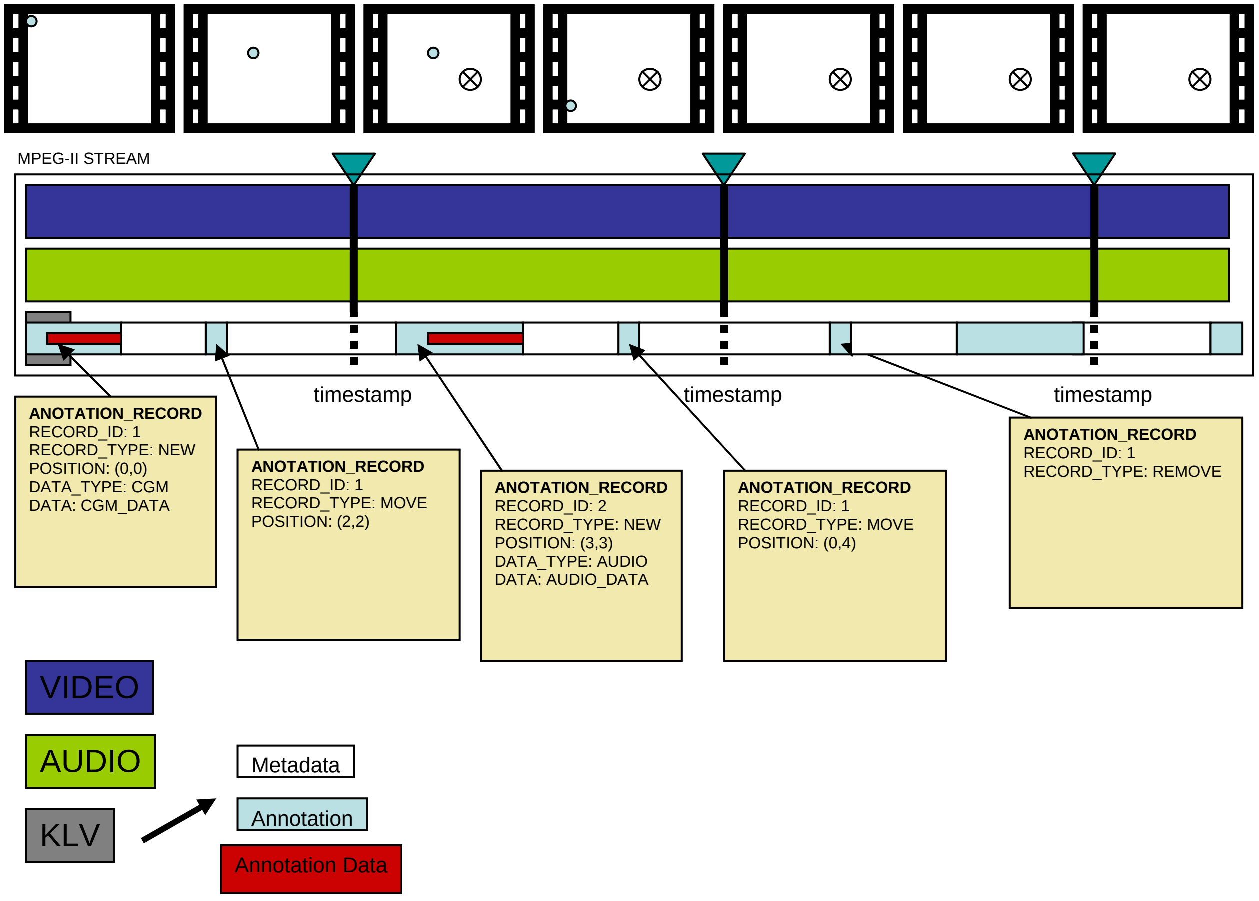


Figure 4 KLV Metadata Embedding

An annotation message with a record type of "NEW" will contain the annotation data, the position on the frame, and the time reference of the frame to indicate when the annotation should be displayed. Annotation messages with record types of "MOVE" will contain the Record ID, position on the screen to move to, and the time reference indicating when to move the annotation. Due to the temporal streaming aspects, video can be opened at an arbitrary time that might not allow the playback to ever encounter a "NEW" command. To compensate for this, the "MOVE" command may also include data that will allow interpreters to render the object correctly. A record type of "REMOVE" will only contain the record ID of the annotation to remove and time when the annotation is to be removed from the screen. Annotation messaging and how annotations can be put in KLV format are further described in Section D.2.1, *Annotations to KLV Format.*

Once annotation messages are created, they are passed to a Message Manager that handles the annotation display. Figure 5 shows how an Annotation Message is processed by the Message Manager. The Message Manager maintains a list of messages. Entries in this list may be added, updated, or removed based on the record type of the message received. If a message is received and the record type or other data is found to be invalid, the message will be discarded.

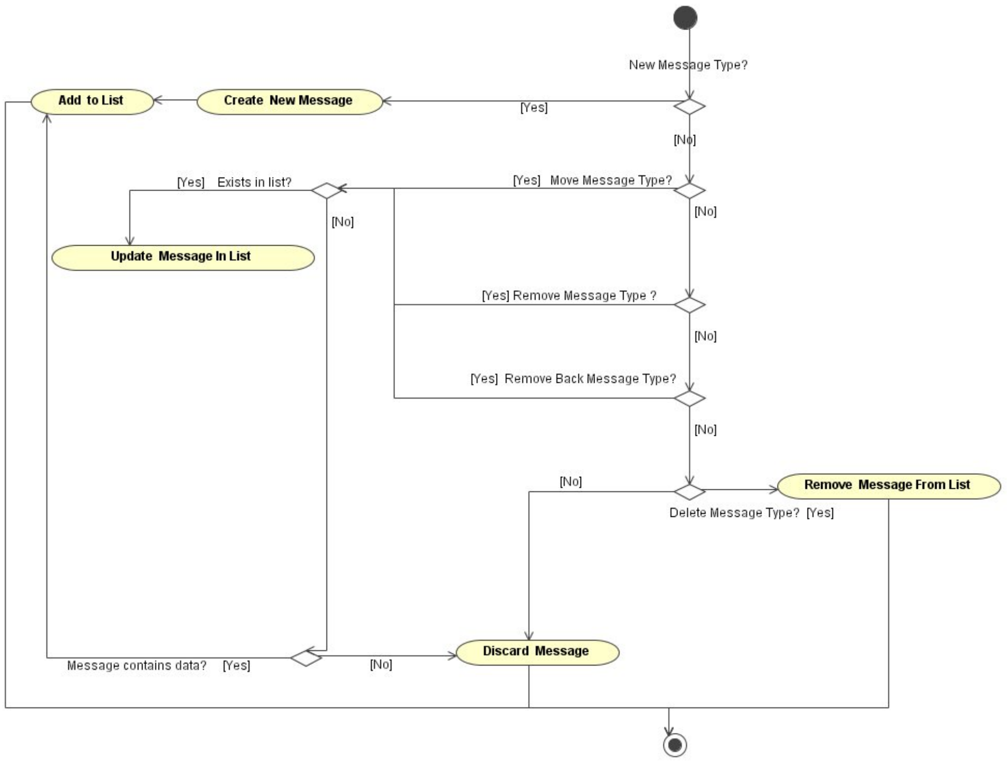


Figure 5 Managing Messages

As the video frames are being displayed, the Message Manager can be called using the frame time reference to display all annotations for that frame. Figure 6 shows how the Message Manager handles requests to display annotations for the desired frame time. The entire list of messages is iterated through, and all relevant annotations are displayed. The annotation will be displayed if the message start time is greater than or equal to the frame time and the end time is less than the frame time.

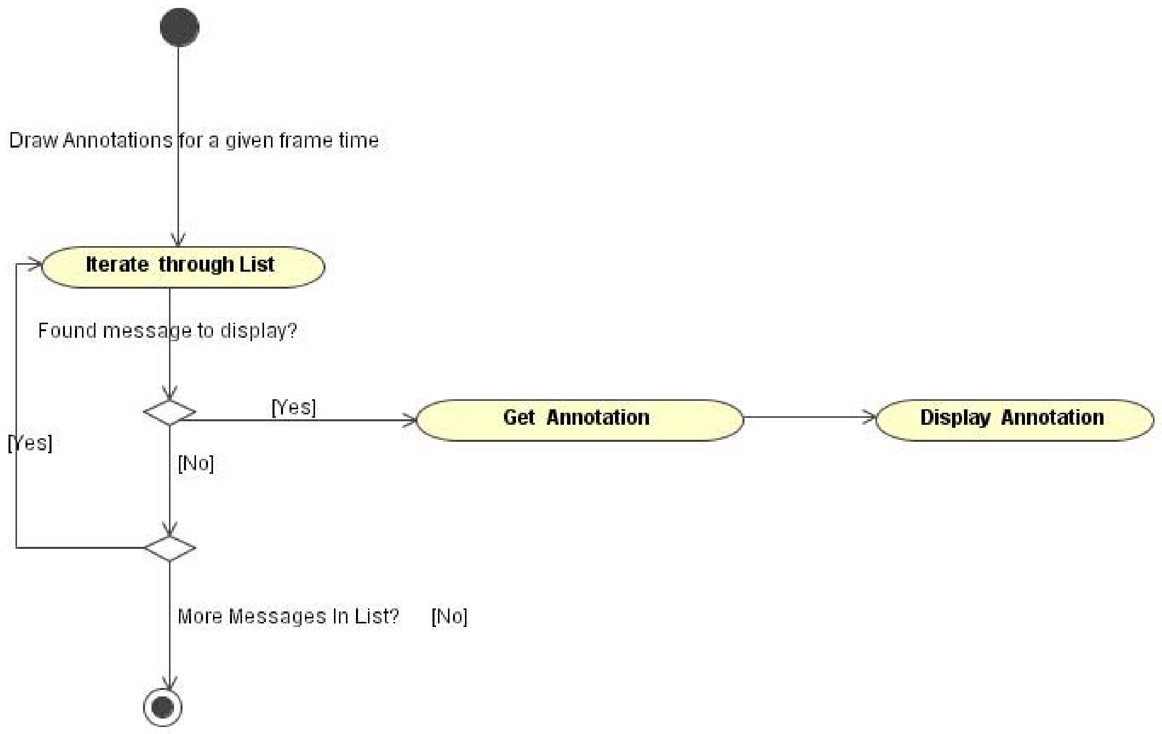


Figure 6 Displaying Annotations

D.2 Annotations in KLV

EXTV requires the ability to recover separate components from an MPEG-2 transport stream of multiplexed video and KLV metadata. Also, the ability to remultiplex interleaved extracted data and new annotation data was desired. Since annotations to a video stream are effectively metadata describing regions of interest or other features, it may be appropriate to insert annotations into an already defined metadata stream format, such as KLV.

D.2.1 Annotations to KLV Format

To format annotation data into KLV, Universal Label (UL) keys were created to define the different fields in an Annotation Message (Table 4). Bytes are numbered from 1 – 16 with lower numbers indicating bytes that occur earlier in the stream.

Every Annotation Message starts with a UL containing a registry category (byte 5 of UL) of 0x02 indicating group data and a registry designator (byte 6 of UL) of 0x01 identifying universal sets, which means entries use the full KLV coding construct throughout and may be arranged in any order.

Table 4 Universal Label (UL) of an Annotation Message

Main UL Header	`0x06,0x0E`
Main UL designators	`0x2B,0x34,0x02,0x01,0x01,0x01`
Main Annotation Item designator	`0x0f,0x01,0x01,0x00,0x00,0x00,0x00,0x00`

The UL of an Annotation Message would be comprised of the Main UL Header, Main UL designators, and Main Annotation Item designator:

`0x06,0x0E,0x2B,0x34,0x02,0x01,0x01,0x01,0x0f,0x01,0x01,0x00,0x00,0x00,0x00,0x00`

The registry category of `0x01` indicates dictionaries, and a registry designator of `0x01` identifies metadata dictionaries.

The required UL keys for an Annotation Message are shown in Figure 7.

KEY		*Length*	**Definition**	Notes
Bytes 1-8	Bytes 9-16			
06 0E 2B 34 02 01 01 01	0F 01 01 00 00 00 00 00	Var.	**Annotation Metadata Set**	
06 0E 2B 34 01 01 01 01	0F 01 02 00 00 00 00 00	2	Record Identity	Unique, non-zero identity
06 0E 2B 34 01 01 01 01	0F 01 03 00 00 00 00 00	2	Record Type	0 = UNDEFINED 1 = NEW 2 = MOVE 3 = REMOVE 4 = REMOVE_BACK 5 = DELETE
06 0E 2B 34 01 01 01 01	0F 01 04 00 00 00 00 00	8	Time Stamp	nanoseconds
06 0E 2B 34 01 01 01 01	0F 01 05 00 00 00 00 00	8	Position	Center x, y display location (4 bytes each)
06 0E 2B 34 01 01 01 01	0F 01 06 00 00 00 00 00	2	Data Type	0 = UNDEFINED 1 = CGM 2 = AUDIO 3 = IMAGE
06 0E 2B 34 01 01 01 01	0F 01 07 00 00 00 00 00	2	Audio Identity	Identity of the sound for this annotation
06 0E 2B 34 01 01 01 01	0F 01 08 00 00 00 00 00	Var.	Data	Raw data
06 0E 2B 34 01 01 01 01	0F 01 09 00 00 00 00 00	Var.	File Type	Mime Type

Figure 7 Annotation UL Description Table

When the application opens a video containing KLV metadata that defines an annotation, annotation messages are created to define the annotations within the system. An annotation message consisting of a unique non-zero Record_Id, Record_Type, Time_Stamp, Position, Data_Type, Audio_Id, and Data will be used by Geo*View for managing annotations. The unique identifier in the message will allow for the identification and tracking of the messages throughout the system. The message will specify the operation being executed on the annotation: NEW, MOVE, or REMOVE. It will include the video time at which the operation (add, move, remove) takes place. The Position gives the center x and y location of the annotation, indicating where on the frame the annotation is to be displayed. Data_Type defines the type of data contained within the message; Audio, Computer Graphics Metafile (CGM), or other data formats. The Audio_Id contains the Id of the associated annotation message that contains the audio to be played when this annotation is first displayed or selected. The Data field is a byte array of the data.

Record_Id, Time_Stamp, and Record_Type are required fields of an annotation message. Record_Id and Record_Type are both integer values, while Time_Stamp is a long value. The Time_Stamp indicates the video time at which the annotation is created, moved, removed, or deleted.

Record_Type values are:

0 – UNDEFINED
1 – new record – requires a unique Record_Id
2 – move record – requires the Record_Id of the annotation to move
3 – remove record – requires the Record_Id of the annotation to remove from this point on in the video
4 – remove back record – requires the Record_Id of the annotation to remove from this point back in the video
5 – delete record – requires the Record_Id of the annotation to be deleted from the video

Optional fields - their field name and t default values are:

Position – x and y both have default values of 0

Data_Type = 0

byte[] Data = null

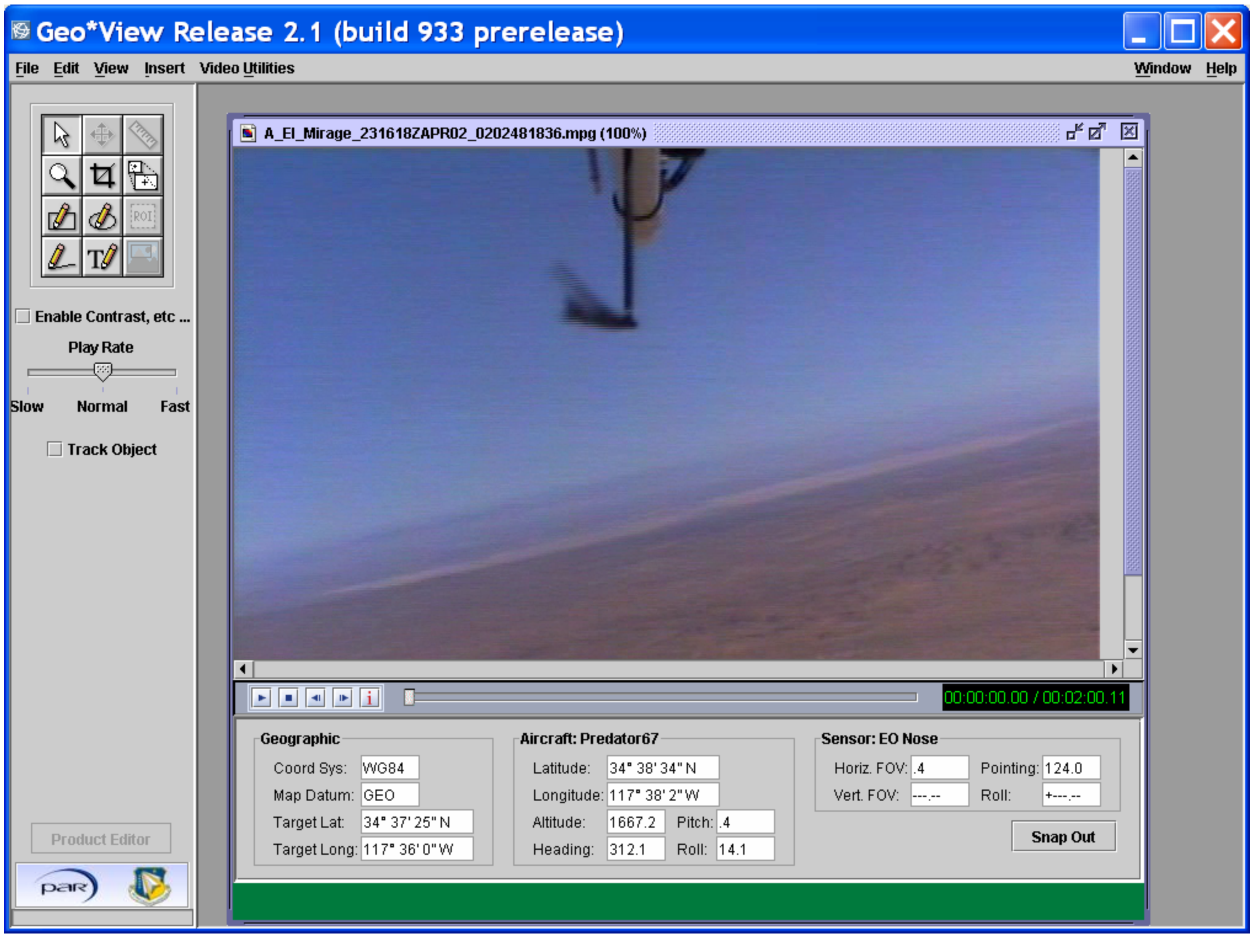
Audio_Id = 0

File_Type = “”

When generating annotations, care should be taken to put the UL groups in sequence based on their Time_Stamps. For example, an annotation needs to be created before it can be moved or removed.

D.3 Displaying KLV metadata

When an MPEG-2 file containing KLV metadata is opened, a metadata display panel will appear below the video display in Geo*View. The metadata display panel may be configured to be part of the Geo*View display (Snapped-In) as shown in Figure 8 or a separate panel that will be displayed in the upper left hand corner of the screen (Snapped-Out) as shown in Figure 9.

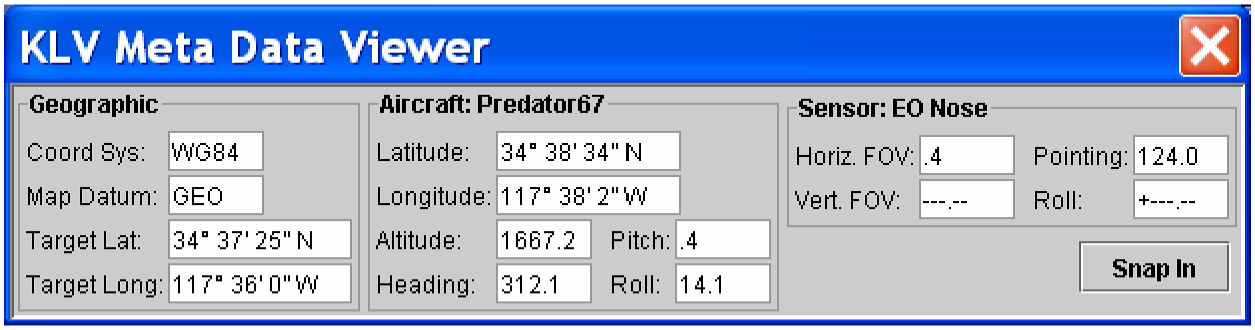

Geo*View Release 2.1 (build 933 prerelease)
File Edit View Insert Video Utilities Window Help
A_El_Mirage_231618ZAPR02_0202481836.mpg (100%)
Enable Contrast, etc ...
Play Rate
Slow Normal Fast
Track Object
Product Editor
00:00:00.00 / 00:02:00.11
Snap Out

Geographic
Coord Sys: WG84
Map Datum: GEO
Target Lat: 34° 37' 25" N
Target Long: 117° 36' 0" W

Aircraft: Predator67
Latitude: 34° 38' 34" N
Longitude: 117° 38' 2" W
Altitude: 1667.2 Pitch: .4
Heading: 312.1 Roll: 14.1

Sensor: EO Nose
Horiz. FOV: .4 Pointing: 124.0
Vert. FOV: ---.-- Roll: +---.--

Figure 8 Snapped-In KLV Display



Geographic
Coord Sys: WG84
Map Datum: GEO
Target Lat: 34° 37' 25" N
Target Long: 117° 36' 0" W

Aircraft: Predator67
Latitude: 34° 38' 34" N
Longitude: 117° 38' 2" W
Altitude: 1667.2 Pitch: .4
Heading: 312.1 Roll: 14.1

Sensor: EO Nose
Horiz. FOV: .4 Pointing: 124.0
Vert. FOV: ---.-- Roll: +---.--

Figure 9 Snapped-Out KLV Display

APPENDIX E – ACRONYMS

480IW	480th Intelligence Wing
ACC/INYR	Air Combat Command/ISR Ground Systems
AFC2ISR/IN	Air Force Command, Control, Intelligence, Surveillance, and Reconnaissance/Intelligence Directorate
AFRL/IFEC	Air Force Research Laboratory/Multi-Sensor Exploitation Branch
API	Application Program Interface
AVI	Audio Video Interleaved
BER	Basic Encoding Rules
CGM	Computer Graphics Metafile
CMIL	Community Motion Imagery Library
CPU	Central Processing Unit
DCGS	Distributed Common Ground Station
DES	Data Extended Segment
DoD	Department of Defense
ELINT	Electronic Intelligence
EXTV	Extended eXploitation Toolkit for Video
GIAS	Geospatial and Imagery Access Services
GOPs	Groups of Pictures
GUI	Graphical User Interface
IR	Infra-Red
IRAD	Independent Research and Development
ISO	International Organization for Standardization
IWG	Interoperability Working Group
JITC	Joint Interoperability Test Command
JMF	Java Media Framework
JSAPI	Java Speech Application Program Interface
KLV	Key Length Value
MISB	Motion Imagery Standards Board
MPEG	Moving Pictures Experts Group
NITF	National Imagery Transmission Format
PES	Packetized Elementary Stream
PGSC	PAR Government Systems Corporation
PID	Packet Identifier

RGB	Red-Green-Blue
ROI	Region of Interest
SAIC	Science Applications International Corporation
SAPI	Speech Application Programming Interface
SMPTE	Society of Motion Picture and Television Engineers
UAV	Unmanned Aerial Vehicles
UL	Universal Label
VPC	Video Processing Capability
WAV	Windows Wave
EXTV	Extended eXploitation Toolkit for Video

APPENDIX F – DOCUMENTS

NITF CGM: MIL-STD-2301A - Computer Graphics Metafile (CGM) Implementation Standard

CGM: ISO/IEC 8632:1999 – Computer Graphics Metafile

FIPS PUB 128-2 - Computer Graphics Metafile (CGM)

MIL-STD-2500A – National Imagery Transmission Format Version 2.0

MIL-STD-2500B – National Imagery Transmission Format Version 2.1

ISO/IEC 13818-1: MPEG-2 Systems

ISO/IEC 13818-2: MPEG-2 Video

SMPTE 336M-2001 – Data Encoding Protocol using Key-Length-Value

ISO/IEC 8825-1 - Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)